Modelling and Verifying an Evolving Distributed Control System Using an Event-Based Approach

Christian Attiogbé

Modelling and Verifying an Evolving Distributed Control System Using an Event-based Approach

Christian Attiogbé

LINA - UMR CNRS 6241 - University of Nantes
F-44322 Nantes Cedex, France
Christian.Attiogbe@univ-nantes.fr

Abstract. Evolving distributed systems are made of several physical devices distributed through a network and a set of functionalities or applications hosted by the physical devices. The configuration of the physical components may be modified through the time, hence the continuous evolving of the whole system. This should affect neither the hosted software components nor the global functionning of the whole system. The components of the systems are software components or physical components but their abstract models are considered with the aim of modelling and reasoning. We show that an event-based approach can be benefically used to model and verify this kind of evolving control systems. The proposed approach is first presented, then the CCTV case study is introduced and modelled. The resulting model is structured as a B abstract machine. The functional properties of the case study are captured, modelled and proved. The refinement technique of Event-B is used to introduce and prove some properties.
Key words: Heterogenous components, Modelling, Event-B, Property verification

1 Introduction

Many software applications are required for decentralized control of highly interacting components; they need to be not only reliable but also extensible hence the use of *evolving distributed control systems*.

Context. The study of distributed systems has been the subject of years of research and development [6, 5, 9]; several results exist at operating systems, middleware and application levels. However specific attention was paid to static distributed environment where the involved hosts (client and server processes or peers), are known and clearly identified as well as their architecture or configuration. Three main architectures have governed the design of distributed applications: two-tier with the interaction between a client and a server, three-tier with two levels of interaction (client-server, server-host) and n-tier architectures. The later is the more general one: a client process located on any host may interact via a middleware with one or several distributed servers.

The client/server two-tier architecture, generalized to n-tier and improved with the peer-to-peer architecture, is used to structure most of the distributed applications.

However, when the environment is not static but dynamic, ie its architecture is evolving according to the (re)configuration of the hosts, then additional difficulties should be considered, more precisely at the system design level. These difficulties rely on the identification of the hosts, the structuring of the exchanged messages, the dynamic aspect of the links and consequently the structure of the overall architecture of the system. Notably, instead of addressing a request to a given server, a thin client will address a request to its environment. There is a shift from the client-server relationship to a more flexible relationship between an application and its virtual environment. Modelling and analysis should not consider the precisely known interacting entities but their virtual counterpart.

Motivation. The current keen interest for virtualized distributed environment (aka *cloud, grid*) pushes a difficulty at application modelling level because, more and more applications are dedicated to *evolving* distributed environment, to store data, to request functionalities or services, etc. Evolving distributed systems are those with an adhoc highly changing architecture, due to the behaviour of their components. Heterogeneity of components (physical devices, software, various models) is a specific feature of these systems. In this work we target such evolving distributed systems and more specifically modelling and reasoning on a system which will be implemented or deployed as an evolving dynamic distributed system. That means the interacting processes are neither explicitly known nor explicitly identified, but the expected behaviour should be described and formally analysed. This is a key for mastering heterogeneity.

Contribution. We propose an event-based approach to make it easy the modelling and the analysis of evolving distributed applications. We show the effectiveness of the proposed approach on a case study: a CCTV[1] control system which may evolve depending on the used devices and their reconfiguration. The approach is based on the use of a virtual network of processes, an event-based modelling and refinement; it can be exploited as a pattern in many other similar cases.

Organisation. The remainder of the article is structured as follows. Section 2 is devoted to the evolving systems considered in the paper; their specificities are underlined and then we present the CCTV case study (Sect. 2.2). Section 3 and Section 4 are the core of the paper. We present the used approach (Sect. 3.2) and its application to the modelling and verification of the CCTV case study (Sect. 4). Finally, Section 5 concludes the article with some lessons and future work.

[1] Closed-Circuit Television

2 Evolving Distributed Systems

2.1 Issues on Evolving Distributed Systems

The correction of distributed systems is still a challenging concern according to modelling, verification and implementation; this is essentially due to mastering complex non-deterministic behaviours, concurrency and safety. Moreover the adoption of the *service-oriented* paradigm enhances not only the need of assistance methods and techniques to build distributed systems but also the needs of highly evolving distributed systems. Processes that represent parts of the system are dynamically linked and unlinked, their behaviours may vary in the time; consequently the global architecture of the system is changing dynamically.

One solution for resolving the difficulties is to build a formal model which serves as a basis for the rigorous analysis and for the construction of (parts of) the system.

Consider a system made of an undefined number of peers which cooperate and provide services; the peers may be mobile, linked and unlinked to different other peers. New peers may be involved with respect to needed functionalities. The architecture of the system is therefore continuously changing.

The message-passing technique with explicit naming and peers identification is not tractable in such an adhoc context; an implicit message passing is needed instead. The evolving should not prevent from maintaining the functionalities and the required properties of a distributed system. Among the issues to be addressed for this purpose, we consider:

- the modelling issue, in order to best capture the requirements and provide a faithful model. Appropriate structuring mechanisms are needed to get an extensible and open system instead of a closed one.
- the reasoning issue, in order to provide method and guidelines to study desired properties. This should be achievable in parts of the global formal model.

We use a real life CCTV case study to present our contribution. This kind of system is often distributed and highly evolving due to the evolution of the architecture of the devices to be controlled.

2.2 The CCTV Control System

A CCTV[2] system is often used for the surveillance of industrial plants, the surveillance of homes or the control of various distributed equipments from a central or a decentralized control room. There are several devices which are linked to their controllers, but this architecture is changing with respect to specific surveillance policies or the adding of new devices, the removing of existing devices. In a basic CCTV system the video captured by cameras are displayed on dedicated screens which may have their own controller.

[2] Closed-Circuit Television

The CCTV system may evolve in such a way that, instead of displaying the video from cameras directly on screens, Digital Video Recorders (DVR) are added to record the video which are used or analyzed later for various purposes (Video Contents Analysis). Consequently, the architecture of system changes: the videos are now recorded before being displayed on the screens.

The system is made of a set of cameras linked with control screens which are under the supervision of humans. The cameras are directed towards parts of a predefined area to be controlled for surveillance or intrusion detection. The cameras send video streams to the control unit which displays the streams on activated screens. More cameras and screens may be added to cover some unreached area without changing the functioning of the system. Thus the CCTV control system is made on the one hand with several controllers, and on the other hand with control screens, control cameras and video display units. The video captured by the cameras are displayed on screens. A controller is linked with one or several (input) cameras, it displays its output on one or several screens, and it may have a DVR for storage.

The requirements stipulate that initially there is no recording of video; then it can be decided to record and save the video while displaying them. Hence a digital video recorder component will be introduced in the system. Accordingly, the video stream will not only be the input of the screens but also the input of the recorder if any.

Some functional properties are required to ensure the good functioning of the control system. We identify and name each of them in order to refer to them in the analysis section. They are summarized in Tab. 1.

FReq_AreaOK	*All the area to be supervised are under control*
FReq_ctrlNCam	*Each controller can manage several cameras*
FReq_cam1Ctrl	*Each camera is managed by only one controller*
FReq_DispOk	*All the captured videos are displayed on some screens*
FReq_RecDispOK	*All active cameras should be under control*
FReq_RecOK	*All video received from the cameras are recorded*
FReq_NewDev	*It should be possible to add new cameras and screens*

Table 1. Synthesis of the functional requirements

We have to model and analyse the functioning of this control system which *i)* is made of several controllers, *ii)* manages several different devices, and *iii)* has a varying architecture.

3 Modelling and Verification Approach

To master the dynamic aspect of evolving systems and the heterogeneity of their components, abstract models with a light composition approach are required.

One solution is to view the systems as a virtual net of components; the components may share abstract channels for communication. At a more concrete level of each component, independently from the other components, the abstract channels may be implemented in various way.

3.1 The Core Modelling Approach

Our approach is based on an event-based composition as a weak coupling of processes that will interact through a common state space that includes abstract channels dedicated to message passing. Our initial work was introduced in [3]. The model of a global system consists of

- a global state space made of the data types identified in the requirements;
- a set of *process types* that describe the identified components of the global system. The component may be physical or logical. We have to identify which are the interacting processes and the messages that they exchange. Each process has its state space, a part of which is shared with the global state space;
- a set of abstract channels to support the communication with the messages that are exchanged among the process types. These channels are part of the global state;
- a set of behavioural descriptions of the process types (they have the form of guarded events).

A control is then handled via the interaction between the components which are modelled as process types. Typically the sense/control/output steps in the standard cycle of control system are handled by the exchanges of messages through the involved components.

3.2 Event-based Global Model: Virtual Net of Interacting Components

We define and link process types via identified common data and abstract channels for interaction (see Fig. 1). The process types are the models of the components identified within the requirements. The abstract channels are modelled according to the interaction needs. Therefore each process type uses the defined abstract channels and state, independently from the other processes; it is the *message-bus* paradigm; here the buses convey messages with data types. Note that this is a refined view which is compliant with the classical approach of modelling a distributed system by a graph whose nodes are the state machines describing processes. In our case the nodes stand for process types, and the abstract channels denote the graph edges and more specifically the interaction means: we have a virtual net of components interacting through the channels.

Therefrom, we have guidelines to help in discovering and modelling the desired behaviours of a system with various architectures, including dynamic ones. We emphasize an event-based view at global level, for composing processes. In

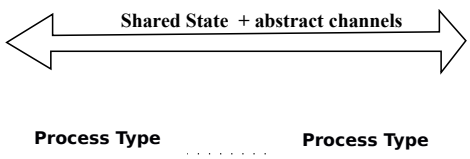

Fig. 1. Virtual net of process types

the description of the behaviour of each process type, only the common abstract channels are used for interaction purpose, enabling thus an independent behaviour with respect to the other processes. They can be of any type, enforcing thus their heterogeneity. Consequently, the architecture of the processes which are connected via the abstract channels is highly flexible. It enables any number of process types, and any number of processes of each type to be composed. In term of distributed control, we can handle in this way the composition of any number of interacting devices and controllers.

For practical experimentation, we use the Event-B notation and method[2]. Event-B is based on first-order logic and set-theory enabling one to use the appropriate mathematical toolkit to capture modelling aspect. To introduce a few notation, total function (denoted by the symbol $\rightarrow$), surjective function (denoted by the symbol $\twoheadrightarrow$) are very useful to express easily some properties as we will see in the modelling and verification part. In the scope of the Event-B method, our process types are modelled as Event-B machines; asynchronous communication is modelled with the interleaved composition of process behaviours which are viewed as event occurrences.

Handling the evolving of architecture. An architecture is the set of processes of various types connected to the abstract channels at a given moment. That is the processes sharing the abstract channels conveying the message passing events from a current configuration. The configuration is submitted to restructuring or changes when the processes evolve.

An instance of a defined process type may join the configuration at any time. In the same way a process may leave a given configuration at any time. These behaviours do not change the modelling of the whole system.

Interaction aspects. Common abstract channels are introduced to link interacting processes and make them communicate. An abstract channel is modelled as a set; we keep it abstract to handle asynchronism. But later in the specification process the channels can be refined, for example as FIFO Queues. The abstract channel is used to wait for a message or to deposit it. The interaction between the processes is then handled using the common abstract channels. Therefore, communications are achieved in a completely decoupled way to favour dynamic structuring. A process may deposit a message in the channel, generating thus an event; other processes may retrieve the message from the channel.

Therefore we use guarded events, message passing and the ordering of event occurrences; the processes synchronise and communicate through the enabling or disabling of their events. An event is used to model the waiting for a data by a process; it may be blocked until the availability of the data which enables the event guard. The availability of the data is the effect produced by another process event. Consider for example the case of processes exchanging messages, one process waits for the message, hence there should be an event with a disabled guard; another process with an enabled guard performs its behaviour which results in sending the message.

Composition of the processes. Practically the composition is implicit during the modelling of the unstructured systems considered here. But a bottom-up view may be adopted, where the composition of process types is made explicit.

The described processes P_i are combined by a *fusion* operation $\biguplus$ that merges an undefined number of process types. The semantics of the fusion operator comes from the conjunction of processes paradigms[10, 1]. The fusion operator merges the state spaces and the events of the processes into a single global system Sys_g which has the conjunction of the invariants, and which in turn can also be involved in other fusion operations.

$$Sys_g \;\widehat{=}\; \biguplus_i P_i \;=\; \biguplus_i \langle S_i, E_i, Evt_i \rangle \;=\; \langle S_g, E_g, Evt_g \rangle$$

According to the fusion operator, when process types are merged, a variable denoting a set of current processes is introduced for each type; this variable is used to identify the processes of the concerned type and to distinguish the events related to each type. The processes access the global state and communicate with others, through their events.

4 Modelling and Verifying the CCTV system

From the requirements we identify the following components: cameras, screens, controllers, DVR. They have specific behaviours, they are loosely linked and their number is varying. We use abstract channels to model the shared communication links (videoChannel, $\cdots$).

The behaviour of a camera is to send a stream of captured signals to the linked screens via the control units (the controllers). The behaviour of a screen consists in visualising the streams of signals received from the cameras. A DVR saves a stream of signal from cameras and also sends them on the screens.

4.1 A Glimpse of the Constructed Model

Following our analysis, the components of the CCTV system, viewed as process types, have been gradually modelled using their weak composition. The result of the composition is a virtual net of processes which is structured using the B notation (with an abstract machine as the structuring unit). As enabled by the

flexibility of the fusion operator used to weakly compose the process types, at the first abstract level, we have combined only the Cameras and the Screens in order to capture earlier the feature imposed by the requirements; it is as if the cameras are linked directly with screens. At a second abstraction level, the controllers are introduced via a refinement where the virtual net is enlarged by other processes; now it is as if the link between cameras and screens is detailed. Using this two abstraction levels, we can handle some properties considering that the policy deployed by the controllers and hence the evolving of the architecture does not impact on the properties to be preserved. This is essentially the initial problem to be solved when considering evolving distributed system. This approach enables us to master the complexity of the model and also to master the verification of the properties. It can be used elsewhere as a modelling and verification pattern.

The structuring of the state space is achieved using identified data types, a set of state variables and an invariant that describes the properties of the processes (camera, screen):

MACHINE *CameraHdl*
...
INVARIANT /* state space predicate */
$connectedCameras \subseteq CAMERA$
/* the set of connected cameras */
$\wedge$ $connectedScreens \subseteq SCREEN$
/* the set of connected screens */
$\wedge$ $activeCameras \subseteq CAMERA$
$\wedge$ $activeScreens \subseteq SCREEN$
$\wedge$ $activeCameras \subseteq connectedCameras$
/* the active cameras are part of the connected ones */
$\wedge$ $activeScreens \subseteq connectedScreens$
$\wedge$ $display \in activeCameras \twoheadrightarrow activeScreens$
/* the active cameras are connected to active screens */
/* All the active screens are used */
$\wedge$ $videoChan \in \mathcal{P}(VIDEO \times CAMERA)$
/* abstract channel: set of video+cameraId */
$\wedge$ $nootherPriority \in BOOL$
$\wedge$ $videoStream \in VIDEO \nrightarrow activeScreens$
/* the video are displayed on one screen */
$\wedge$ $displayedVideo \in \mathcal{P}(VIDEO)$
$\wedge$ $activeDVR \subseteq DVR$
/* the set of active DVR */
$\wedge$ $videoStore \in VIDEO \nrightarrow activeCameras$
/* to store the video from the cameras */
$\wedge \cdots$ /* more properties are added below */

Fig. 2. Abstract model of the CCTV system

The evolving of the system depends on the behaviour (modelled as a set of events) that defines the involved processes. The events considered for the Camera model description are summarised in Tab. 2.

Behaviour related to Camera	
Event	Description
addCamera	a new camera is added
activateCamera	one camera is activated
sendVideo	a camera sends a video
rmvCamera	a camera is removed

Table 2. Camera handling events

Behaviour related to Screen	
Event	Description
addScreen	a screen is added
getVideo	a screen gets a video
displayVideo	a screen displays a video
rmvVideo	a screen is removed
addDVR	a DVR is added
getVideoDVR	a DVR gets a video

Table 3. Screen handling events

The B specification to manage a Camera is the abstract machine equipped with these related events (see Fig. 3). In the same way the behaviour dedicated to the screen control is depicted in Fig. 4.

4.2 Mastering the Architecture and its Modelling

One of the advantages of the composition of the process types by the fusion operator is that an important part of the model can be incrementally analysed, by considering each process type, whatever the order. Not all the behaviours can be analysed this way due to the lack of information for the interaction between the processes; but as soon as the appropriate types are introduced the interaction analysis is achieved. From the composition point of view it is very beneficial for mastering the evolving architecture. The reconfiguration of the system architecture for example does not impact on the modelling and the control of the system. However there are some limitations to this type of composition: one can neither constrain the composition nor hide some communication channels. From the heterogeneity point of view, data abstraction and behavioural abstraction

MACHINE *CameraHdl*
SETS CAMERA, SCREEN, VIDEO
VARIABLES
connectedCameras, *activeCameras*, *display*, *videoChan*, *nootherPriority*
activeDVR, *videoStore*
INVARIANT
/* state space predicate, as given above (Fig. 4.1) */
INITIALISATION
connectedCameras, *activeCameras*, *display*, *videoChan*,
nootherPriority, *activeDVR*, *videoStore* := $\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$
EVENTS
`addCamera` $\widehat{=}$ ...
; `activateCamera` $\widehat{=}$ ...
; `sendVideo` $\widehat{=}$ ...
; `rmvCamera` $\widehat{=}$ ...
END

Fig. 3. Structure of the Camera abstract machine

MACHINE *ScreenHdl*
SETS SCREEN, VIDEO /* abstract sets */
VARIABLES
connectedScreens, *activeScreens*, *display*, *videoChan*, *nootherPriority*,
videoStream, *displayedVideo*, *activeDVR*, *videoStore*
INVARIANT
/* state space predicate, as given in Fig. 4.1 */
INITIALISATION
connectedScreens, *activeScreens*, *display*, *videoChan*, *nootherPriority*,
videoStream, *displayedVideo*, *activeDVR*, *videoStore* := $\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$
EVENTS
`addScreen` $\widehat{=}$ ...
; `getVideo` $\widehat{=}$ ...
; `displayVideo` $\widehat{=}$ ...
; `rmvVideo` $\widehat{=}$ ...
; `addDVR` $\widehat{=}$ ...
; `getVideoDVR` $\widehat{=}$ ...
; ...
END

Fig. 4. Structure of the screen abstract machine

allow to consider in the same way heterogeneous components via their process types.

We do not deal with the parameterisation of the architecture in this case study.

4.3 Verifying the properties

The requirements stipulate that the system may satisfy the properties introduced in the table Tab.1 (see Sect. 2.2). Some of them are captured without restructuring. Some others are restructured; for instance, the requirement FReq_RecDispOK is rephrased as follows: all active cameras are recorded and displayed when the DVR is activated.

Properties Descriptions
FReq_DispOK: *All the captured videos are displayed on some screens.*
FReq_RecDispOK: *When the DVR is installed, all the displayed video are recorded and save to ensure the DCA.*
FReq_RecOK: *All active camera are recorded and displayed*

We have completely modelled and analyzed the system using the presented approach, including the properties formalized (in Event-B) as follows.

The property FReq_RecDispOK is modelled as follows:

$$((activeDVR \neq \{\} \land videoStore \neq \{\}) \Rightarrow ((\mathsf{dom}(videoStore) \subseteq displayedVideo) \lor (\mathsf{dom}(videoStore) \setminus (displayedVideo \cap \mathsf{dom}(videoStore)) \subseteq \mathsf{dom}(videoStream))))$$

The set inclusion is used in this formalisation to capture the FReq_DispOK property.

The properties FReq_cam1Ctrl and FReq_DispOK are captured through a total surjective function: $display \in activeCameras \twoheadrightarrow activeScreens$.
Indeed the domain ($activeCameras$) of the total function indicates that all the active cameras are displayed. The property FReq_RecOK is captured in the invariant with the same total surjective function (because all the active cameras that are displayed on the screens are recorded when the VCR is used).

The property FReq_NewDev is handled through the events (*addCamera*, *activateCamera*) of the machine *CameraHdl* and the event *addScreen* of the *ScreenHdl* machine; they impact on the variable *display*.

The abstract machine corresponding to the composition of the process types is obtained by the merging of the parts of the machines of the processes. The resulting machine is named IntergratedVideoSys. This later is refined in the following.

Refinement and verification The properties FReq_ctrlNCam and FReq_cam1Ctrl are captured through a refinement (named IntergratedVideoSys_r1) of the abstract machine modelled previously.

To master the modelling and the verification of the required properties we use the refinement technique available in the Event-B method. During the analysis of the case, one can note that the three properties FReq_DispOk, FReq_ctrlNCam and FReq_cam1Ctrl are dependent and we model them gradually. Indeed from an abstract point of view the FReq_DispOk property is captured with a single total function *display*, expressing that each camera (video) is displayed on one screen and all the videos are displayed. Thus in the first abstract machine we do not introduce the properties FReq_ctrlNCam and FReq_cam1Ctrl; they are introduced in a refinement.

The refinement we have used is summarized as follows: a function $f : A \nrightarrow B$ is refined by two functions g and h defined using the same sets A, B together with a new set I such that:

$$g : A \rightarrow I \ \wedge \ h : I \nrightarrow B \wedge$$
$$f \subseteq (g;\ h)$$

More specifically, I stands for the set of controllers which have been introduced in the refinement; the total function g and the surjecive function h are used to capture respectively the properties FReq_cam1Ctrl and FReq_ctrlNCam.

Consequently, the three properties are proved using the refinement of the previous abstract machine. This refinement process is a practical modelling and verification pattern easily reusable for similar cases of control involving controllers and controlled units in this decentralized way.

The abstract machine and its refinement have been implemented using the Rodin toolkit (see Fig. 5 for the synthetic architecture).

The proof statistics (with the refinement related to the properties FReq_DispOk, FReq_ctrlNCam and FReq_cam1Ctrl) obtained from the Rodin toolkit are drawn in the following table Tab. 4 . The context machines have their names suffixed by `Ctx`, they contain the definitions of sets which are shared by the other machines of the B project. The context machines do not generate proof obligations.

All the proof obligations for the correctness of our models (containing the required properties) have been automatically discharged by the Rodin toolkit.

4.4 Further with the interoperability

A key to handle heterogeneity and semantic interoperability is the use of a layered structure composed of formal models, where the inner layer, the most abstract one, is the most commonly homogeneous in terms of concepts, relations and properties. The outermost layers are those with more specific details in the models. Note that the proposed method should be viewed from the abstract layer

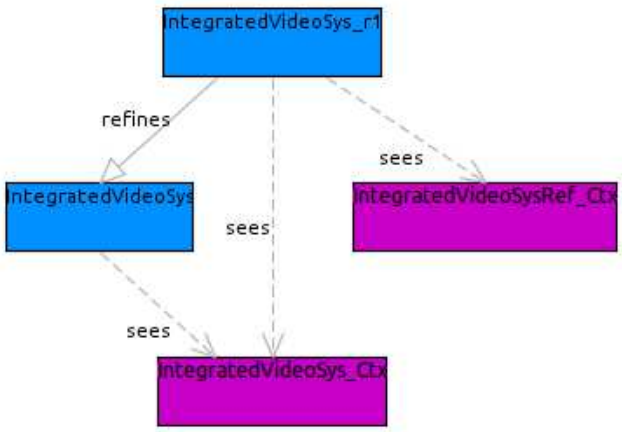

Fig. 5. Structure of the models (a snapshot from the Rodin toolkit)

ElementName	Total PO	Auto proved	Undischarged PO
IVS_CCTV (project)	30	30	0
IntegratedVideoSysRef_Ctx	0	0	0
IntegratedVideoSys_Ctx	0	0	0
IntegratedVideoSys	24	24	0
IntegratedVideoSys_r1	6	6	0

Table 4. Proof statistics

which is one of the many levels needed to master heterogeneity and semantic interoperability. Indeed, from a low abstraction level, several formal descriptions with various semantic models may be associated to one given component of a system. However changing the abstraction level to the higher one, details are forgotten until one can reach an abstract level where the semantics models are interoperable; this corresponds to the event level and the virtual component net level adopted in the current proposed work. The global properties and their analysis are only possible at this level.

Establishing bridges between formal models, using for instance the matching between domain specific ontological concepts is necessary to deal with intermediary abstraction layers. A reference compatibility layer is required as the commonly shared semantics; this is the smallest set of properties shared by all components of a system. As far as the implementation is concerned, this reference compatibility layer is captured by the invariant of our Event-B model. For this purpose, the choice of Event-B is worthwhile since Logics and Set Theory have very basic concepts which can be easily shared or implemented with other

formalisms. Likewise, when we have to consider the composition of components modelled with different formalisms, on the one hand the bridging between the formalisms and on the other hand the reference to a compatibility layer are the key solutions.

Therefore a tool specific analysis is required when different formal models are considered to tackle different facets of a system. When it is necessary and possible, equivalence proofs should be conducted but this is not required for all the different facets. Interfaces between the models have to be built even with the vision of narrowing or widening the models and their coverage.

In approaches such as Ptolemy [7, 8] or ModelHex [4], the compatibility between computation models or their synchronisation are emphasized; this is like a semantic adaptation in order to make the models compatible. The main difference between these approaches and our is that we do not achieve a semantic adaptation since we consider that the heterogeneity is inherently a feature of complex systems. Rather we try to handle heterogeneity but by maintaining consistency between the involved components.

5 Conclusion

We have presented a method to model and analyse a distributed control system with a varying architecture. The method is based on the composition of the identified physical and logical components of the system described at the abstract level. The components are described as process types. Their composition is based on the sharing of abstract channels denoting message buses, used by the processes to communicate without the identification of the interacting peers. This enables us to handle the distributed and dynamic architecture of the control system. The method helps to structure and model interacting components as process types. To put into practice, the Event-B notation and method are used. It follows that the refinement technique permits to handle some properties via refinement of abstract structure defined at earlier steps. Especially, in order to handle the independence of the control with respect to the controllers and their architecture, the desired relationships between the controlled units are established at abstract level as if they are directly linked; then controllers are introduced in a refinement and appear between the linked controlled units. All the properties are proved either at abstract level or in the refinement. Experimentations are conducted with a CCTV case study using AtelierB and Rodin. Our experiment is easily reusable in other case studies involving a control system with a dynamic architecture. The two-levels specification can be considered as the reference pattern when we distinguish firstly at the abstract level the end-point relationship between the controlled units (ie of the desired control policy) and secondly the structuring of the controllers and their link as intermediary agents between the controlled units. Further works are scheduled on the parametrisation of the architecture of the controllers. In the current case of the CCTV the structuring constraints are fixed. However, in some control systems for instance in embedded control systems, in order to manage energy consumption, the structure of the system can

be reconfigured. This leads us to think about various behavioural modes and to define the structuring and thus the architecture with respect to these modes. The desired properties and hence their proofs will depend on the behavioural modes. But this can be as well analysed as various refinements of the same abstract machine of the ongoing architecture.

References

1. M. Abadi and L. Lamport. Conjoining Specifications. *ACM Trans. Program. Lang. Syst.*, 17(3):507–535, May 1995.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
3. C. Attiogbé. Event-Based Approach to Modeling Dynamic Architecture: Application to Mobile Adhoc Network. In T. Margaria and B. Steffen, editors, *ISOLA'2008*, volume 17 of *CCIS*, pages 769–781, 2008.
4. F. Boulanger, C. Jacquet, C. Hardebolle, and A. Dogui. Heterogeneous model composition in modhel'x: the power window case study. In *Proceedings of Gemoc 2013, Workshop on the Globalization of Modeling Languages*, page 10 pages, Miami, Florida, USA, Sep 2013.
5. L. Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.
6. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.
7. E. A. Lee. Disciplined Heterogeneous Modeling. In O. H. D.C. Petriu, N. Rouquette, editor, *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering, Languages, and Systems (MODELS)*, pages 273–287. LNCS 6395, Springer-Verlag, October 2010.
8. C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.
9. A. S. Tanenbaum and R. van Renesse. Distributed Operating Systems. *ACM Comput. Surv.*, 17(4):419–470, 1985.
10. P. Zave and M. Jackson. Conjunction as Composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.